

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

PARAZITNÍ KAPACITY PŘI ŘEŠENÍ ELEKTRICKÝCH OBVODŮ

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

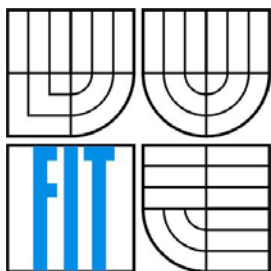
AUTOR PRÁCE
AUTHOR

MICHAL KADÁK

BRNO 2007



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

PARAZITNÍ KAPACITY PŘI ŘEŠENÍ ELEKTRICKÝCH OBVODŮ

ELECTRONIC CIRCUITS SIMULATIONS AND PARASITIC CAPACITANCES

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE

AUTHOR

MICHAL KADÁK

VEDOUCÍ PRÁCE

SUPERVISOR

DOC. ING. JIŘÍ KUNOVSKÝ, CSC.

BRNO 2007

Zadání bakalářské práce

Řešitel: **Kadák Michal**

Obor: Informační technologie

Téma: **Parazitní kapacity při řešení elektrických obvodů**

Kategorie: Teorie obvodů

Pokyny:

1. Seznamte se s problematikou numerického řešení obyčejných diferenciálních rovnic s přímým využitím Taylorovy řady a jejich řešením v TKSL.
2. Vypracujte metodiku řešení elektrických obvodů s parazitními kapacitami s využitím techniky diferenciálního počtu.
3. Proveďte analýzu stability numerického řešení, počtu rovnic a složitosti matematických úprav.
4. Výpočty ověřte v TKSL.
5. Srovnajte výsledky se světovými standardy.
6. Metodické postupy aplikujte při návrhu programového vybavení pro automatické generování soustavy ekvivalentních diferenciálních rovnic.

Literatura:

- Dle zadání vedoucího

Při obhajobě semestrální části projektu je požadováno:

- 1,2

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese
<http://www.fit.vutbr.cz/info/szz/>

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním paměťovém médiu (disketa, CD-ROM), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Kunovský Jiří, doc. Ing., CSc., UITS FIT VUT**

Datum zadání: 1. listopadu 2006

Datum odevzdání: 15. května 2007

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav inteligentních systémů
602 00 Brno, Božetěchova 2

doc. Dr. Ing. Petr Hanáček
vedoucí ústavu

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami

1. Pan

Jméno a příjmení: **Michal Kadák**

Id studenta: 84326

Bytem: Mateja Bela 2504/17, 911 08 Trenčín

Narozen: 26. 07. 1985, Trenčín

(dále jen "autor")

a

2. Vysoké učení technické v Brně

Fakulta informačních technologií

se sídlem Božetěchova 2/1, 612 66 Brno, IČO 00216305

jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....

(dále jen "nabyvatel")

Článek 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
bakalářská práce

Název VŠKP: Parazitní kapacity při řešení elektrických obvodů

Vedoucí/školicitel VŠKP: Kunovský Jiří, doc. Ing., CSc.

Ústav: Ústav inteligentních systémů

Datum obhajoby VŠKP:

VŠKP odevzdal autor nabyvateli v:

tištěné formě počet exemplářů: 1

elektronické formě počet exemplářů: 2 (1 ve skladu dokumentů, 1 na CD)

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2

Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti:
 - ☒ ihned po uzavření této smlouvy
 - ☐ 1 rok po uzavření této smlouvy
 - ☐ 3 roky po uzavření této smlouvy
 - ☐ 5 let po uzavření této smlouvy
 - ☐ 10 let po uzavření této smlouvy(z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3

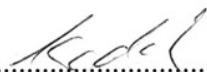
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísní a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne:

.....

Nabyvatel


.....

Autor

Abstrakt

Práca sa zaoberá použitím parazitných kondenzátorov pri výpočte elektrických obvodov pomocou diferenciálnych rovníc, ktoré sa použijú ako vstup do simulačného nástroja TKSL, umožňujúceho výpočet rozsiahlych systémov diferenciálnych rovníc metódou Taylorovej rady. Druhá časť sa zaoberá návrhom a implementáciou grafickej nadstavby nad TKSL, ktorá bude kresliť elektrické obvody.

Kľúčové slová

TKSL, Taylorov rad, grafický editor, kreslenie vodičov, parazitné kondenzátory, diferenciálne rovnice, .NET

Abstract

This work deals with using parasitic capacitances to solve electronic circuits and generate differentials equations as input to TKSL. TKSL is simulation language which allows you to solve large systems of differentials equations. The second part deals with design and implementation of graphic editor, which allows you to draw electronic circuits.

Keywords

TKSL, Taylor series, graphic editor, line drawing, parasitic capacitances, differentials equations, .NET

Citácia

Michal Kadák: Parazitné kapacity pri riešení elektrických obvodov, bakalárska práca, Brno, FIT VUT v Brně, 2007

Parazitné kapacity pri riešení elektrických obvodov

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením Doc. Ing. Jiřího Kunovského, CSc.

Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Michal Kadák
12.4. 2007

PodĎakovanie

Na tomto mieste by som chcel poďakovať svojmu školiťovi, Doc. Ing. Jiřímu Kunovskému, CSc. za odbornú pomoc, ktorú mi pri realizácii tejto práce poskytol.

© Michal Kadák, 2007.

Táto práca vznikla ako školské dielo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práca je chránená autorským zákonom a jej užitie bez udelenia oprávnenia autorom je nezákonné, s výnimkou zákonom definovaných prípadov..

Obsah

Obsah	1
Úvod	3
1 TKSL	4
1.1 Vývoj TKSL/386	4
1.2 Taylorova metóda	4
1.3 Použitie TKSL/386	6
2 Parazitné kondenzátory pri riešení elektrických obvodov	8
2.1 Klasická metóda riešenia jednoduchého obvodu	8
2.1.1 Transformácia trojuholník - hviezda	8
2.1.2 Výpočet klasickou metódou	9
2.2 Použitie parazitných kondenzátorov	11
2.3 Príklad v TKSL	13
2.4 Stiff systémy	15
3 Návrh grafického editoru	16
3.1 Požiadavky na grafický editor	16
3.2 Analýza	16
3.2.1 Kreslenie vodičov	16
3.2.2 Automatické vytváranie uzlov	17
3.2.3 Uloženie informácií o obvode	17
3.3 Návrh	17
3.3.1 Mriežka	18
3.3.2 Kolmé kreslenie vodičov	19
3.3.3 Uloženie informácií o obvode – databáza	21
4 Implementácia	22
4.1 Platforma a vývojové prostredie	22
4.2 .NET	22
4.3 Implementácia Databázy	23
4.3.1 PIN	23
4.3.2 CID	23
4.4 Implementácia vykresľovaných tvarov	24
4.5 Implementácia mriežky	25
4.6 Implementácia kresliacich nástrojov	26
5 Použitie	27
5.1 Zapojenie modulu do programu	27

5.2	Manuál.....	27
6	Záver	29
	Literatúra	30
	Príloha A.....	31

Úvod

Existuje mnoho spôsobov ako riešiť elektrické obvody. Jedným z nich sú klasické metódy, ktorými nás učia počítať už na stredných školách. Sú to obvykle pracné a zdĺhavé výpočty, ktoré nás síce dovedú ku správne riešeniu, ale nie vždy cestou, ktorou by sme chceli ísť. Preto sa zaoberajme druhým spôsobom riešenia a to riešenia pomocou diferenciálnych rovníc.

Riešenie pomocou diferenciálnych rovníc je elegantné a priamočiare. V dnešnej dobe viaceré komerčné systémy na výpočet elektrických obvodov používajú diferenciálny počet a vzniknuté sústavy diferenciálnych rovníc riešia pomocou bežných numerických metód.

Na našej fakulte sa už veľa rokov vyvíja systém TKSL (TKSL/C), ktorý rieši rozsiahle systémy diferenciálnych rovníc pomocou metódy Taylorovho radu. Vývojom tohto systému, jeho použitím a príkladmi sa zaoberá kapitola 1.

Pri zostavovaní diferenciálnych rovníc niekedy nastávajú problémy a môže sa zdať, že diferenciálne rovnice sa zostrojiť nedajú. A preto sa vyvíja teória pripojenia kondenzátorov do obvodu, tak aby zmeny spôsobené kondenzátorom neboli znateľné ale aby nám jeho zapojenie vyriešilo všetky väčšie problémy s výpočtom. Táto vznikajúca teória nesie názov metóda parazitných kondenzátorov a popisuje ju kapitola 2.

Aktuálna verzia TKSL má na vstupe zoznam diferenciálnych rovníc a konštánt. Ďalší krok vývoja je navrhnuť a implementovať grafickú nadstavbu, v ktorej by sa nakreslilo schéma obvodu a automaticky by sa vygenerovali diferenciálne rovnice pre TKSL. Návrhom takého editoru sa zaoberá kapitola 3. Jeho implementácia je konkrétne popísaná v kapitole 4. Kapitola 5 popisuje zavedenie modulu do programu a jeho použitie.

1 TKSL

1.1 Vývoj TKSL/386

Simulačný jazyk TKSL/386, ktorý je predchodcom systému TKSL/C, bol vytvorený pre testovanie algoritmov využívajúcich k riešeniu (systémov) diferenciálnych rovníc a ostatných problémov Taylorovho radu. Tento systém bol vytvorený v užívateľsky priateľskom prostredí TurboVision. Dovoľuje užívateľovi nastaviť presnosť výpočtu, rád metody. Zabezpečuje presnú detekciu nespojitostí, výpočet prebieha s premenným integračným krokom.

Výhodou systému je možnosť použitia schémy nakreslenej v systéme ORCAD. Ďalšia možnosť je priame využitie integrovaného editoru zdrojového kódu pre prekladač. Systém TKSL/386 disponuje prekladačom jazyka vhodného pre jednoduchý popis diferenciálnych rovníc.

Po úspešnom preložení kódu popisujúceho analyzovaný systém je možné spustiť simuláciu. Priebeh riešenia sú prehľadne zobrazené v grafe, pomocou kurzoru sa dá zobrazit' hodnota riešenia vo vyznačených bodoch.

1.2 Taylorova metóda

Majme rovnicu a hľadáme jej riešenie:

$$y' = f(x, y), y(x_0) = y_0$$

Povedzme, že úloha je numericky vyriešená, práve keď sa podarí zostaviť riešenie v tvare:

$$y(x_0 + h) \doteq y(x_0) + \Delta y_0(x_0, y_0, h)$$

Odvedenie Taylorovho polynómu

Idea odvedenia Taylorovho vzorca je nasledujúca. Je daná funkcia f a bod a taký, že existuje okolie $H_a \subset D_f$ a existuje $f^{(n)}(a)$. Budeme vytvárať polynóm P tak, aby $P(a) = f(a)$, ..., $P(a) = f^{(n)}(a)$, t.j. aby polynóm P a funkcia f mali v bode a styk n -tého rádu. Otázkou je, či taký polynóm vôbec môže existovať, ak existuje, tak či je určený jednoznačne, ako presne aproximuje pôvodnú funkciu apod. Na tieto otázky mám dajú odpovede nasledujúce vety. V nasledujúcich konštrukciách použijeme konvenciu: funkčnú hodnotu funkcie v bode budeme považovať za hodnotu nultej derivácie funkcie v tom bode (t.j. $f(a) = f^{(0)}(a)$).

Veta 1.2 *Nech pre funkciu f , bod $a \in R$ a číslo $n \in N$ platí:*

$$(1) (\exists H_a)(H_a \subset D_f),$$

$$(2) \text{ existuje } f^{(n)}(a) \in R.$$

Potom existuje práve jeden polynóm $T_{(n)}$ najvyššieho stupňa n tak, že platí:

$$(\forall k \in \hat{n} \cup \{0\})(T_n^{(k)}(a) = f^{(k)}(a)).$$

Dôkaz: Nech P je polynóm stupňa najviac n , $P(x) = \sum_{k=0}^n a_k (x-a)^k$. Chceme nájsť

koeficienty a_k tak, aby:

$$(\forall j = 0, \dots, n)(P^{(j)}(a) = f^{(j)}(a)). \quad (1.2)$$

Je ale:

$$P^{(j)}(x) = \sum_{k=j}^n k \dots (k-j+1) a_k (x-a)^{(k-j)}$$

odkiaľ:

$$P^{(j)}(a) = j! a_j.$$

Teda podmienka 1.2 vedie na sústavu rovníc:

$$\begin{aligned} a_0 &= f(a) \\ a_1 &= f'(a) \\ &\vdots \\ a_j &= \frac{1}{j!} f^{(j)}(a) \\ &\vdots \\ a_n &= \frac{1}{n!} f^{(n)}(a) \end{aligned}$$

To je ale sústava $n+1$ rovníc pre $n+1$ neznámych s diagonálnou (teda jednotkovou) maticou. Z Frobeniovej vety je zrejmé, že táto sústava má jediné riešenie.

Poznámka 1.2 *Polynóm $T_{(n)}$ z vety 1.2 má teda tvar:*

$$T_n(x) = \sum_{k=0}^n \frac{1}{k!} f^{(k)}(a)(x-a)^k$$

Ak urobíme Taylorov rozvoj funkcie y , získame

$$y(x_0 + h) - y(x_0) = hy'(x_0) + \frac{h^2}{2} y''(x_0) + \dots$$

Ďalej platí

$$y'(x_0) = f(x_0, y_0) = f_0$$

Tento vzťah môžeme ďalej derivovať na:

$$\begin{aligned} y''(x_0) &= \frac{\partial f}{\partial x}(x_0, y_0) + \frac{\partial f}{\partial y}(x_0, y_0)y'(x_0) = f_x + f_y f_0 \\ y'''(x_0) &= \frac{\partial^2 f}{\partial x^2}(x_0, y_0) + 2 \frac{\partial^2 f}{\partial x \partial y}(x_0, y_0)y'(x_0) + \\ &+ \frac{\partial^2 f}{\partial y^2}(x_0, y_0)y'^2(x_0) + 2 \frac{\partial f}{\partial y}(x_0, y_0)y''(x_0) = \\ &= f_{xx} + 2f_{xy}f_0 + f_{yy}f_0^2 + f_y(f_x + f_y f_0) \end{aligned}$$

A tak sa dá pokračovať ľubovoľne dlho (za predpokladu, že f má derivácie dostatočne vysokého rádu). Použitie Taylorovho radu sa javí ako veľmi rýchle a veľmi presné, aj keď je Taylorov rad považovaný za nevhodný, hlavne z dôvodu nutnosti získavania vyšších derivácií.

1.3 Použitie TKSL/386

Predpokladajme, že máme riešiť rovnicu s nulovými počiatočnými podmienkami:

$$y''' + a_2 y'' + a_1 y' + a_0 y = b_3 z''' + b_2 z'' + b_1 z' + b_0 z \quad (1.3)$$

Simulátor, analogicky s metodikou numerického riešenia diferenciálnych rovníc, akceptuje iba výrazy s prvou deriváciou, je najskôr treba rovnicu 1.3 previesť na ekvivalentnú sústavu diferenciálnych rovníc prvého rádu. Môžeme použiť metódu postupnej integrácie a tým získame sústavu rovníc pre TKSL:

$$\begin{aligned} A' &= b_0 z - a_0 y && \& 0 \\ B' &= b_1 z - a_1 y + A && \& 0 \\ C' &= b_2 z - a_2 y + B && \& 0 \\ y &= b_3 z + C && \& 0 \end{aligned}$$

Vstupom pre systém TKSL/386 bude nasledujúci kód:

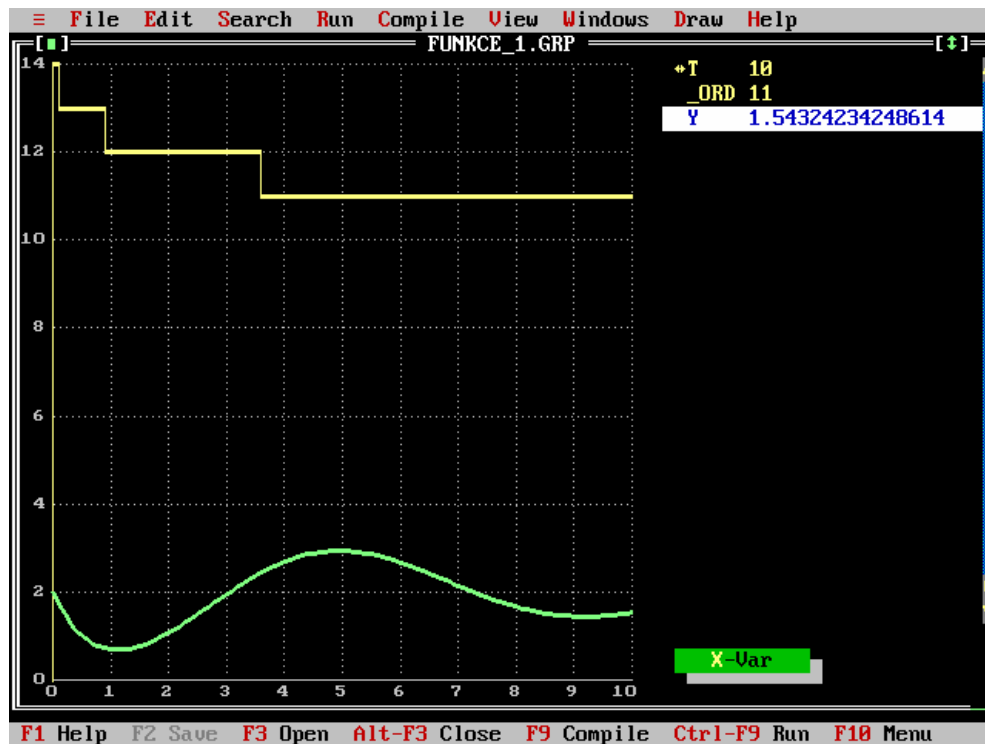
```
var
y,z,A,B,C; { deklarácia premenných }
const
b0=2, b1=1, b2=1, b3=2,          { definície konštánt }
a0=1, a1=1, a2=2,
tmax = 10, DT=0.1, EPS = 1e-20;
system
  A'= b0*z - a0*y &0;           { zápis jednotlivých rovníc }
  B'= b1*z - a1*y + A &0;       { za & je počiatočná podmienka }
  C'= b2*z - a2*y + B &0;
```

```

y = b3*z + C;
z = 1;
sysend;
xaxis(0,10);
yaxis(0,5).

```

Na obrázku 1.3 je zobrazený výsledok simulácie predchádzajúcej rovnice v programe TKSL/386. Lomená čiara navrchu vyjadruje krok Taylorovho radu, ktorý bol použitý v danej časovej jednotke na výpočet. Krivka v dolnej časti vyjadruje hodnoty premennej y počas priebehu simulácie.



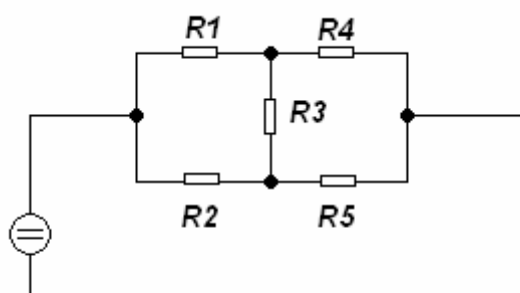
Obrázok 1.3: Výstup simulácie programu TKSL

2 Parazitné kondenzátory pri riešení elektrických obvodov

Existuje veľké množstvo klasických postupov ako vyriešiť elektrické obvody zadané schémou. V tejto kapitole sa zameriame na výpočet elektrického obvodu klasickou metódou a následne použitím diferenciálnych rovníc a parazitných kapacít.

Z dôvodu názornosti a prehľadnosti zvolíme jednoduchý obvod, ktorý sa bude skladať len z odporov a zdroja napätia. Dôležité bude zapojenie odporov, ktoré sa bude musieť pri použití klasických metód prepočítavať.

Majme elektrický obvod zadaný nasledovnou schémou:

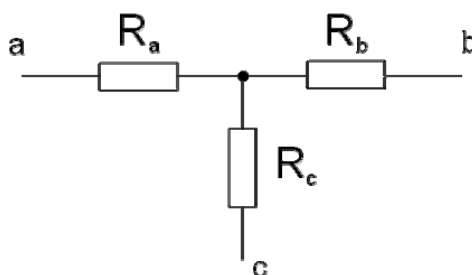


Obrázok 2.0: Schéma jednoduchého obvodu

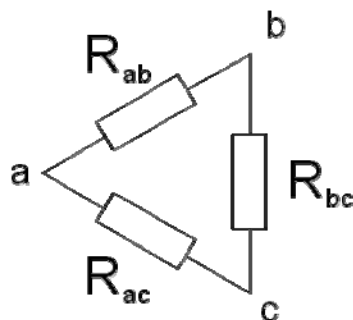
2.1 Klasická metóda riešenia jednoduchého obvodu

2.1.1 Transformácia trojuholník - hviezda

Niekedy sa pri riešení sietí stretneme s odporovým trojpólom tvaru hviezdy alebo trojuholníka, viz. obr. 2.1a 2.1b.



Obrázok 2.1a: Hviezda



Obrázok 2.1b: Trojuholník

Tieto trojpóly sú navzájom zameniteľné a často takouto zámenou môžeme docieľiť zjednodušenie siete. Označíme si vrcholy trojuholníka a odpovedajúce vrcholy hviezdy písmenami a, b, c . Odporov jednotlivých ramien hviezdy označíme R_a, R_b, R_c podľa príslušnosti ramena k vrcholu. Podobne označíme odporov ramien trojuholníka R_{ab}, R_{bc}, R_{ac} . Skúsme spočítať odporov medzi vrcholmi a, b hviezdy. Tento odpor sa spočíta ako $R_a + R_b$. U trojuholníka je tento odpor rovný paralelnej kombinácii odporu R_{ab} s odporom daným súčtom $R_{ac} + R_{bc}$. Vzhľadom na to, že chceme nahradiť hviezdu trojuholníkom, prípadne trojuholník hviezdou, musí sa odpor medzi vrcholmi a, b u hviezdy rovnať odporu medzi vrcholmi a, b u trojuholníka, čím dostávame prvú rovnicu. Podobne pre vrcholy b, c a c, a dostaneme ďalšie dve rovnice, ktoré môžeme riešiť buď pre neznáme R_a, R_b, R_c alebo pre neznáme R_{ab}, R_{ac}, R_{bc} . Dostaneme tieto vzťahy:

$$R_{ab} = \frac{R_a R_b + R_b R_c + R_c R_a}{R_c} \quad (2.1.1a)$$

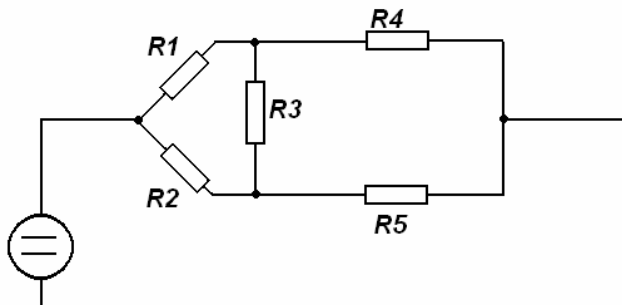
a analogické dva, ktoré sa dajú popísať nasledovne: Odpor strany trojuholníka je rovný súčtu 3 možných súčinov odporov vetví hviezdy, ktorý sa podelí odporom protiľahlej strany hviezdy. Pre hviezdu dostaneme:

$$R_a = \frac{R_{ab} R_{ca}}{R_{ab} + R_{bc} + R_{ca}} \quad (2.1.1b)$$

2.1.2 Výpočet klasickou metódou

Pri prvom pohľade na obvod z obrázku 2.0 sa nám môže javiť riešenie veľmi jednoduché a priamočiare. Stačí spočítať veľkosti odporov podľa známych vzorcov a vypočítať výsledný odpor celého obvodu. Pozrime sa však na obvod trochu detailnejšie. Problém nám robí odpor (R_3), ktorý je zapojený vertikálne a spája vrchnú vetvu so spodnou. Tento odpor nám bráni v jednoduchom spočítaní celkového odporu.

Pri druhom pohľade si môžeme všimnúť, že zapojenie ľavej časti je vlastne zapojenie do trojuholníka. Toto je názorne vidieť na obrázku 2.1.2a.

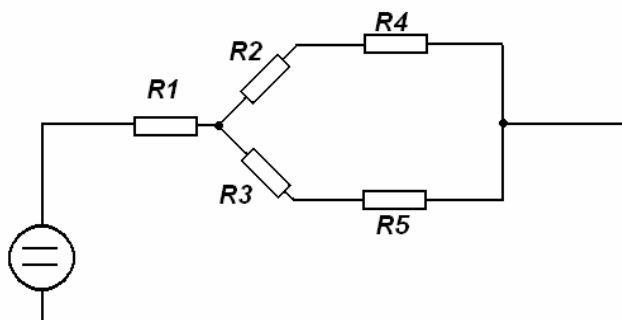


Obrázok 2.1.2a: Zapojenie do trojuholníka

Vzhľadom na toto zistenie bude ďalším krokom riešenia transformácia trojuholníkového zapojenia na hviezdu. Podľa vzorca 2.1.1b získame vzorec na výpočet jednotlivých nových odporov v zapojení do hviezdy (obrázok 2.1.2b). Konkrétny vzorec na výpočet novej hodnoty odporu R_1^* je:

$$R_1^* = \frac{R_1 R_2}{R_1 + R_2 + R_3}$$

analogicky si môžeme odvodiť vzorce na výpočet ďalších dvoch nových odporov.



Obrázok 2.1.2b: Transformovaný obvod na hviezdu

Po transformácii sme už dostali schému, v ktorej sa nenachádza vertikálne zapojený odpor ak na začiatku a teda môžeme začať upravovať známymi vzorcami. Odpor R_2 a R_4 sú zapojené do série, teda ich môžeme spočítať a tým získame odpor $R_{2,4}$. Rovnakým spôsobom spočítame aj sériovo zapojené odpory R_3 a R_5 . Získali sme dva paralelne zapojené odpory $R_{2,4}$ a $R_{3,5}$. Podľa vzorca pre spočítanie dvoch paralelne zapojených odporov spočítame predposledný odpor $R_{2,3,4,5}$.

Vzorec pre sčítanie dvoch paralelne zapojených odporov, vychádza zo vzťahu:

$$\frac{1}{R} = \frac{1}{R_1} + \frac{1}{R_2}$$

Znamená to teda, že prevrátená hodnota výsledného odporu sa rovná súčtu prevrátených hodnôt oboch paralelne zapojených odporov. Takže vzorec pre náš výpočet po úprave je:

$$R_{2,3,4,5} = \frac{R_{2,4} R_{3,5}}{R_{2,4} + R_{3,5}}$$

Po tomto kroku nám v obvode zostali už len dva, do série zapojené, odpory $R_{2,3,4,5}$ a R_1 . Keďže sú v sérii, tak ich môžeme jednoducho spočítať a tým získame celkový odpor obvodu. Výpočet prúdu, ktorý preteká obvodom je potom jednoduchý. Požijeme Ohmov zákon (2.1.2) a pomocou celkového odporu a veľkosti napätia vypočítame tečúci prúd.

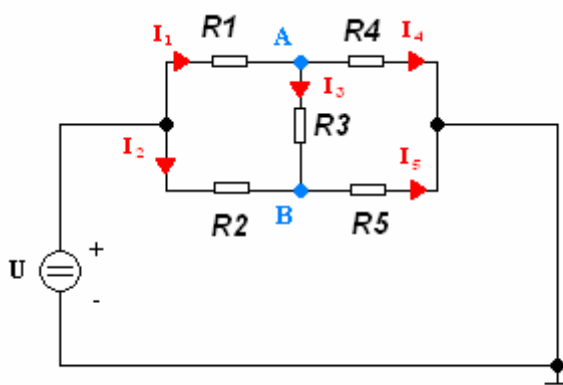
$$I = \frac{U}{R} \quad (2.1.2)$$

2.2 Použitie parazitných kondenzátorov

Ukázali sme si postup riešenia pomocou klasických metód. Ten istý obvod teraz vyriešime pomocou diferenciálnych rovníc a parazitných kondenzátorov.

Obvod je rovnaký ako v predchádzajúcom prípade, je neupravený transformáciou na hviezdu, teda stále máme zapojený vertikálny odpor R_3 . Výhoda použitia parazitných kondenzátorov je, že nepotrebujeme upravovať obvod transformáciou. Potrebujeme len nájsť uzol, v ktorom by sme chceli vedieť napätie. Presnejšie je to uzol, ktorého napätie nám pomôže vypočítať všetky hodnoty obvodu, ktoré chceme.

V našom obvode sme si vybrali uzly (obrázok 2.2a), ktorými je pripojený práve ten odpor, ktorý nám bráni jednoduchému riešeniu, teda R_3 . Samozrejme je možné zvoliť všetky uzly v obvode. Ale nie je treba počítať uzly, ktoré nakoniec nebudeme potrebovať k výpočtu. Preto je dôležité sa zamyslieť a vybrať vhodne.



Obrázok 2.2a: uzly

Pri znalosti napätí v uzloch A a B sa dá použiť metóda uzlových napätí, ktorou sa nám podarí vypočítať prúdy tečúce jednotlivými vetvami. Metóda vychádza z 1. Kirchhoffovho zákona, ktorý znie: Súčet všetkých prúdov do uzla vstupujúcich sa rovná súčtu prúdov z uzla vystupujúcich.

Konkrétne využitie metódy uzlových napätí je vidieť na rovniciach pre výpočet prúdov v našom obvode:

$$I_1 = \frac{U - U_A}{R_1}$$

$$I_2 = \frac{U - U_B}{R_2}$$

$$I_3 = \frac{U_A - U_B}{R_3}$$

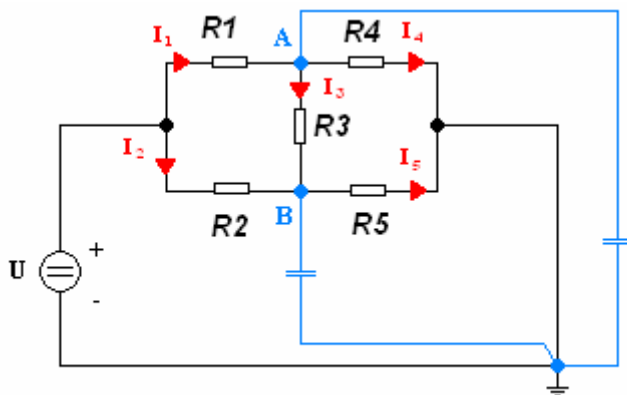
$$I_4 = \frac{U_A - 0}{R_4}$$

$$I_5 = \frac{U_B - 0}{R_5}$$

Ako je vidieť, tak prúd v konkrétnej vetve sa vypočíta ako rozdielu napätia uzlu, z ktorého vyteká a uzla, do ktorého následne vteká. A tento rozdiel sa podelí odporom na konkrétnej vetve. V posledných dvoch rovniciach vidíme, že napätie uzla, do ktorého prúd vteká má pravdepodobne hodnotu 0. Je to spôsobené, tým, že obvod je uzemnený a tieto dva prúdy vtekajú do tejto uzemnenej časti, ktorá má nulové napätie.

Ukázali sme si, že pri znalosti napätia na uzloch A a B sme schopný vypočítať prúdy vo všetkých vetvách. Pre výpočet týchto napätí použijeme parazitné kondenzátory.

Parazitné kondenzátory sa pripoja na nami zvolené uzly a uzemnia sa (obrázok 2.2b). Pri zopnutí obvodu sa parazitné kondenzátory začnú nabíjať a po určitom čase sa kondenzátory nabijú a prúd vo vetve kondenzátora prestane tečť. Tento stav nazývame ustálený. Napätie na kondenzátore v tomto stave je napätie, ktoré potrebujeme pri ďalšom výpočte.



Obrázok 2.2b: Parazitné kondenzátory

Pre výpočet napätí na uzloch si vytvoríme diferenciálne rovnice. Diferenciálna rovnica pre výpočet napätia na kondenzátore je:

$$U' = \frac{1}{C} I$$

V našom prípade budú rovnice pre výpočet vyzerat' takto:

$$U_A' = \frac{1}{C} I_A$$

$$U_B' = \frac{1}{C} I_B$$

Prúdy I_A a I_B sú prúdy tečúce vo vetvách s pridanými kondenzátormi, čo je vidieť na obrázku 2.2b. Pre ich výpočet použijeme 1. Kirchhoffov zákon. Rovnice budú vyzerat' nasledovne:

$$I_1 - I_3 - I_4 = I_A$$

$$I_2 + I_3 - I_5 = I_B$$

V tejto fáze už máme všetky potrebné informácie a všetky rovnice sú zostavené. Stačí určiť hodnoty odporov a napätie na zdroji a s týmito veličinami vyriešiť vzniknuté diferenciálne rovnice. Pre tento účel použijeme aplikáciu TKSL.

2.3 Príklad v TKSL

V predchádzajúcej časti sme si odvodili rovnice pre výpočet. V tejto časti si ukážeme, či boli naše teórie pravdivé a to tak, že odsimulujeme priebeh nami vytvorených diferenciálnych rovníc v systéme TKSL. Podľa výstupných grafov sa ukáže, či kondenzátory pracujú ako sme predpokladali.

Zvoľme teda hodnoty odporov a to tak, že $R_1 = 200 \, \Omega$, $R_2 = 300 \, \Omega$, $R_3 = 500 \, \Omega$, $R_4 = 100 \, \Omega$, $R_5 = 50 \, \Omega$ a vstupné napätie bude $U = 32V$.

Potom po dosadení do vzorcov v časti 2.2 získame tento zdrojový kód pre systém TKSL:

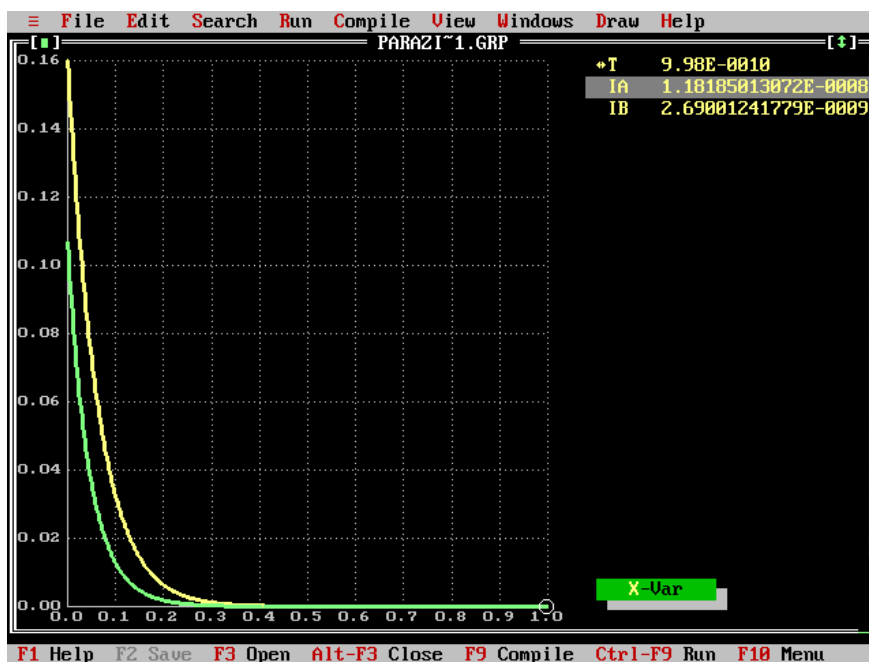
```
var i1, i2, i3, i4, i5, iA, iB, uA, uB;
const
u = 32, R1 = 200, R2 = 300, R3 = 500, R4 = 100, R5 = 50, C = 1e-12,
eps=1e-20, Tmax=1e-9, dt=1e-12;
system
    iA=i1-i3-i4;
    iB=i2+i3-i5;
    uA'=1/C * iA &0;
    uB'=1/C * iB &0;
    i1= 1/R1 * (u-uA);
    i2= 1/R2 * (u-uB);
    i3= 1/R3 * (uA-uB);
```

```
i4= 1/R4 * (uA-0);
i5= 1/R5 * (uB-0);
sysend.
```

Po spustení simulácie nám TKSL vykreslí graf, v ktorom názorne uvidíme ako sa jednotlivé hodnoty menili v čase. Na obrázku 2.3a je znázornený priebeh napätí na kondenzátoroch. Je vidieť, že napätia na kondenzátoroch sa na začiatku postupne nabíjali a po uplynutí určitého času sa ustália na stabilných hodnotách.



Obrázok 2.3a: Priebeh napätí na kondenzátoroch



Obrázok 2.3b: Priebeh prúdov vo vetvách kondenzátorov

Na obrázku 2.3b je znázornení priebeh hodnôt prúdov vo vetvách kondenzátorov. Podľa simulácie je vidieť, že vo vetvách tiekol prúd, kým sa kondenzátory nenabíli a neustálil sa stav. Po tomto okamihu sa prúdy minimalizovali a vetvami prestali prúdy tečť.

Ako posledné je na obrázku 2.3c zobrazený priebeh prúdov v obvode, na ktorom máme znázornené jednotlivé hodnoty prúdov. V čase kedy nastal ustálený stav sú vypočítané hodnoty v jednotlivých vetvách.



Obrázok 2.3c: Prúdy v obvode

Z grafu sa dajú prečítať hodnoty vypočítaných uzlov v jednotlivých vetvách. Zvládli sme to, čo sme v časti 2.1 robili klasickými metódami a transformáciami, použitím diferenciálnych rovníc a parazitných kondenzátorov. Táto metóda je veľmi presná a rýchla a poskytuje nám možnosť riešiť aj zložitejšie schémy elegantnou cestou diferenciálnych rovníc.

2.4 Stiff systémy

Použitie parazitných kondenzátorov má aj svoje problémy. Ak chceme pripojiť k obvodu parazitný kondenzátor, tak by mal byť dostatočne malý, teda mal by mať oproti prvkom v obvode o niekoľko rádov menšiu hodnotu kapacity. Dôvod je aby zmenil chovanie obvodu len tak minimálne aby sa dalo zanedbať. A to je problém pri riešení takej diferenciálnej rovnice, v ktorej sú nelineárne prvky, ktorých parametre sa líšia až v niekoľkých rádoch.

Takéto diferenciálne systémy potrebujú na vyriešenie prvkov s nízkou hodnotou veľmi malý krok pri riešení pomocou Taylorovho radu. Na druhej strane sú tam prvky s vysokou hodnotou, ktoré budú mať dlhý simulačný čas. Takže v konečnom dôsledku bude simulácia trvať veľmi dlhú dobu. Také systémy nazývame STIFF systémami.

3 Návrh grafického editoru

V prvej kapitole sme sa zaoberali simulačným nástrojom TKSL. Ukázali sme si, že s jeho pomocou dokážeme jednoducho spočítať zložité diferenciálne rovnice. V druhej kapitole sme si ukázali, akým spôsobom môžeme vytvoriť diferenciálne rovnice, opisujúce konkrétny obvod. Je vidieť, že vytvorenie diferenciálnych rovníc je trochu komplikované a zdĺhavé. Preto bola navrhnutá grafická nadstavba nad simulačným nástrojom TKSL.

Tento grafický vstup umožňuje nakresliť schéma elektrického obvodu a toto schéma v určitom tvare poskytnúť ďalšiemu modulu, ktorý ho transformuje na diferenciálne rovnice.

3.1 Požiadavky na grafický editor

Hlavné požiadavky na navrhovaný editor sú:

- Graficky príjemne a elektrotechnicky presné znázornenie prvkov
- Možnosť kreslenia základných súčiastok
 1. cievka
 2. kondenzátor
 3. odpor
 4. zdroj
- Ortogonálne (kolmé) kreslenie vodičov spájajúcich komponenty
- Možnosť automatického vytvorenia uzlov pri dotyku s vodičom
- Ukladanie informácií o obvode v tvare, ktorý bude poskytnutí ďalej

3.2 Analýza

V súčasnosti existuje mnoho grafických editorov na kreslenie elektrických obvodov. Za zmienku stojí systém Simulink a TKSL/C Lab, čo je aplikácia vyvinutá na našej fakulte. Pri analýze a návrhu som vychádzal hlavne z týchto dvoch systémov.

3.2.1 Kreslenie vodičov

Je zložité vytvoriť užívateľsky príjemné prostredie, v ktorom bude možné kresliť spojovacie čiary, teda vodiče, nielen rýchlo, ale aj rovno a teda kolmo na seba. Problémov okolo kreslenia vodičov postupom času a realizácie systému vyplynulo viac, najpodstatnejšie sú:

- Napojenie vodiča na vstupe či výstupe komponenty
- Napojenie vodiča doprostred existujúceho vodiča

- Úprava vodičov pri presunoch komponent
- Po odstránení vodiča je nutné odstrániť pripojenie komponent spojených týmto vodičom

Prvé dva problémy rieši Simulink jednak algoritmom zalamovania vodičov a jednak systémom mriežky. Je oveľa jednoduchšie umiestniť koniec vodiča na viacbodovú oblasť, než sa pokúšať umiestniť koniec na jeden konkrétny bod.

Je dobré si uvedomiť, že takto sa vodič sám zalamuje. Pri napojovaní výstupu je orientácia vodiča opačná. Týmto prístupom je veľmi zrýchlené samotné kreslenie vodičov, čo má za následok užívateľský komfort. Veľmi podobne je tiež riešené napojenie vodičov doprostred existujúcej čiary, tu existujú štyri možnosti zalomenia čiary.

Tretí problém je riešený tak, že sa automaticky vkladajú alebo odstraňujú z čiary body tam, kde dochádza buď ku krivosti (čiaru na sebe nie sú kolmé), alebo kde sú tri body na jednej rovnnej čiare.

3.2.2 Automatické vytváranie uzlov

Ďalšou podmienkou je aby sa automaticky vytvárali uzly pri umiestnení konca vodiča na existujúci vodič. Pri tomto kroku narazíme na určité problémy:

- Pri umiestnení konca vodiča na existujúci vodič je treba vytvoriť uzol
- Na mieste kde sme vytvorili uzol je potrebné rozdeliť existujúci vodič na dva nové vodiče
- Komponenty, ktoré boli doteraz spojené jedným vodičom už nie sú spojené s ním ale s novovzniknutým uzlom
- Pri vymazaní uzla je nutné zmazať všetky vodiče do neho vchádzajúce

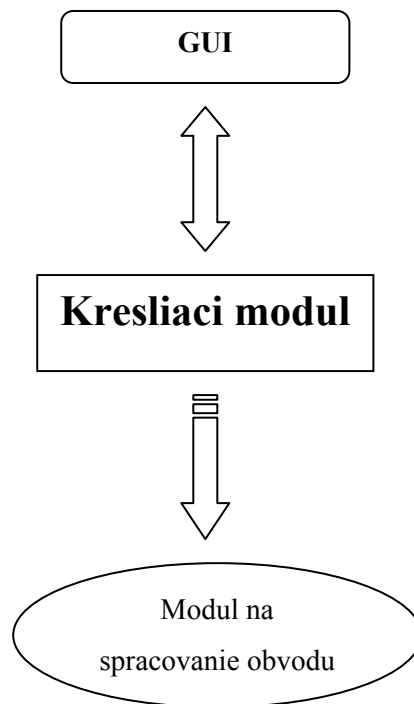
3.2.3 Uloženie informácií o obvode

Informácie o aktuálne nakreslenom obvode by mali byť sprostredkované ďalšiemu modulu prípadne modulom na spracovanie. Preto nastal problém, ako uložiť podobu schémy obvodu, tak aby obsahovala len dôležité informácie a bola jednoducho prístupná.

Samozrejme manipulácia s takouto štruktúrou by mala byť jednoduchá a rýchla, pretože počas kreslenia sa do nej bude veľmi veľa krát pristupovať. Vždy keď sa zmení akákoľvek hodnota komponenty, musí nastať zmena hodnoty v štruktúre.

3.3 Návrh

Pri návrhu grafického editora som vychádzal z požiadaviek v kapitole 3.1 a analýzy z kapitoly 3.2. Zameral som sa na fakt, že editor momentálne neobsahuje žiadne prívetivé grafické rozhranie, ale je to len modul nad obsluhou správ nad plátnom. Teda je úplne oddelené samotné kreslenie od grafického rozhrania, čiže od GUI.



Obrázok 3.3: Návrh komunikácie kresliaceho modulu s GUI a Modulom na spracovanie obvodu

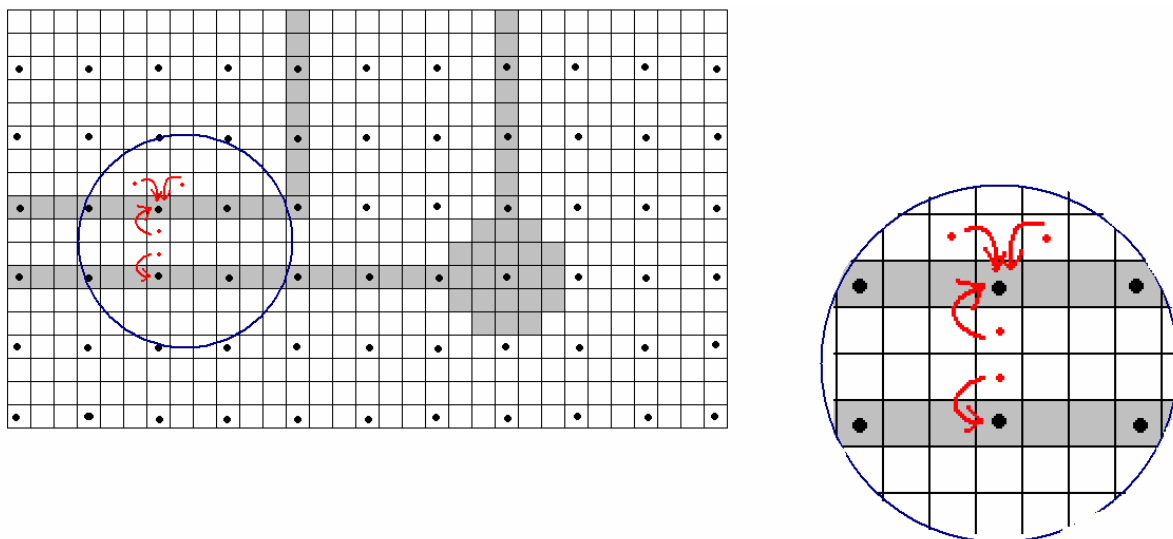
Na obrázku 3.3 je vidieť, že kresliaci modul nie je závislý na grafickom užívateľskom rozhraní a už vôbec nie na iných moduloch spracovania. Komunikácia medzi GUI a kresliacim modulom pracuje na základe udalostí. Ak nastane udalosť myši tak GUI jednoducho odovzdá parametre tejto udalosti kresliacemu modulu. Ten túto správu vyhodnotí a vykreslí, čo bude potreba, prípadne prepíše určité hodnoty v štruktúre informácií o obvode.

Komunikácia smerom od kresliaceho modulu ku GUI je vlastne len žiadosť o prekreslenie plochy, ktorú posielal kresliaci modul podľa vlastných potrieb.

3.3.1 Mriežka

Ako bolo spomenuté v kapitole 3.2.1, je pre užívateľa príjemnejšie snažiť sa umiestniť koniec vodiča na určitú množinu bodov ako na jeden jediný. Pomocou mriežky rozdelíme kresliacu plochu, ktorej jeden bod je jeden pixel, na väčšie časti. Zvoľme teda mriežku tak aby bola dostatočne jemná a neobmedzovala užívateľa v kreslení, ale musí byť hrubšia ako najmenší rozlíšiteľný bod.

Optimálne rozloženie mriežky je 3 pixely. To znamená, že medzi susednými bodmi mriežky budú práve dva pixely. Je to výhodné z dôvodu, že ak sa klikne na ľavý pixel z tejto dvojice tak je jednoduché vypočítať, že kliknutý bod v mriežke bude práve ten vľavo. Analogicky, ak klikneme na pravý pixel z dvojice, v mriežke to bude práve ten bod v pravo. Je jasné, že medzi dvoma pixelmi už nie je žiadny, na ktorý sa dá kliknúť. Takže je toto rozloženie exaktné. Konkrétne situáciu mriežky ukazuje obrázok 3.3.1.



Obrázok 3.3.1: Mriežka a transformácia bodov

Z obrázka je jasné, že všetky vodiče budú ležať na bodoch mriežky. Komponenty už nemôžeme nakresliť tak jednoducho ako vodiče, takže jednotlivé body komponent môžu a budú zasahovať mimo body mriežky. Ale je veľmi dôležité aby ich vstupy a výstupy boli zarovnané presne na body mriežky.

Pri dodržaní umiestnenia dôležitých bodov na mriežku už nebude problém jednoducho pripájať vodiče na vstupy a výstupy komponent. Je jasné, že pri pohybe nakreslenou cievkou alebo odporom, sa takýto objekt bude umiestňovať nie podľa aktuálnej pozície myši, ale jeho súradnice sa prepočítajú na mriežku.

3.3.2 Kolmé kreslenie vodičov

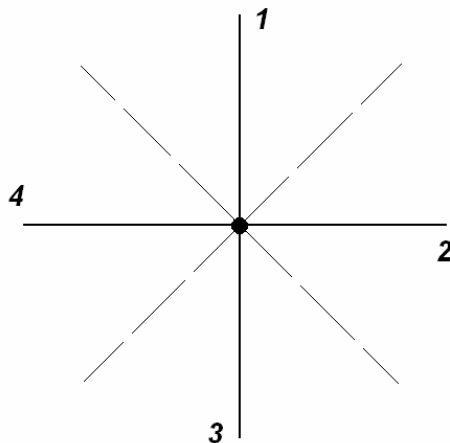
Charakteristickou črtou všetkých editorov elektrických obvodov je, že vodiče sú na seba kolmé. Pri bližšom pohľade na tento problém si uvedomíme, že riešenie nebude triviálne. Musíme si stanoviť ako moc flexibilne sa nám jednotlivé vodiče budú zalamovať. Ja som zvolil len jedno zalomenie momentálne kreslenej čiary. Neznamená to, že by nakreslený vodič mohol byť zalomený len raz. Znamená to, že pri aktuálnom ťahaní časti vodiča sa táto časť zalomí len raz.

Postup takého kreslenia vodiča bude nasledovný:

- Užívateľ klikne na vstup komponenty
- Pri ťahaní myšou sa mu bude zobrazovať segment vznikajúceho vodiča, ktorý sa bude maximálne v jednom bode zalamovať
- Pri kliknutí na kresliacu plochu sa nový segment zapíše ako súčasť vodiča a ťahá sa ďalší segment rovnakým spôsobom
- Pri kliknutí do vstupu alebo výstupu inej komponenty sa kreslenie vodiča ukončí

Po tomto algoritme nám vznikne ľubovoľne mnohokrát zalomení vodič. Ktorý bude presne navrhnutý podľa potrieb užívateľa.

Rozhodovací algoritmus, ktorý má posúdiť akým smerom sa bude vodič kresliť je založený na nasledujúcich vzťahoch:



Nech bod A je začiatkový bod segmentu a bod B je koncový bod segmentu, potom platí:

- pre prvý prípad:

$$\begin{aligned} |X_B - X_A| &< |Y_B - Y_A| \\ Y_B - Y_A &> 0 \end{aligned}$$

- pre druhý prípad

$$\begin{aligned} |X_B - X_A| &> |Y_B - Y_A| \\ X_B - X_A &> 0 \end{aligned}$$

- pre tretí prípad

$$\begin{aligned} |X_B - X_A| &< |Y_B - Y_A| \\ Y_B - Y_A &< 0 \end{aligned}$$

- pre štvrtý prípad

$$\begin{aligned} |X_B - X_A| &> |Y_B - Y_A| \\ X_B - X_A &< 0 \end{aligned}$$

Pomocou predchádzajúcich vzťahov sme schopný jednoducho vypočíta súradnice bodu, v ktorom nastane zalomenie. Tým pádom vieme aj smer, akým má viesť vodič z posledného bodu jeho kreslenia.

3.3.3 Uloženie informácií o obvode – databáza

Schéma elektrického obvodu je grafická záležitosť a na jej uchovanie v pamäti a jej distribuovanie ďalej je potreba nejakým spôsobom ju uložiť vo forme čísel a písmen. Preto som navrhol štruktúru, ktorá bude obsahovať údaje o obvode a nazval som ju **databáza**.

Nie je to databáza v pravom slova zmysle, je to len štruktúra, konkrétnejšie trieda napísaná v programovacom jazyku. Táto trieda v sebe obsahuje kolekciu komponent obvodu (cievky, odpory, kondenzátory), kolekciu vodičov v obvode a kolekciu uzlov v obvode.

Každý element v databáze má svoje unikátne identifikačné číslo a obsahuje položky, ktoré sú pre daný element potrebné.

Element komponenty obsahuje samozrejme svoje identifikačné číslo a ďalej tieto položky:

- typ komponenty (či je to cievka, odpor, kondenzátor alebo zdroj)
- dva PINy, teda vstup a výstup komponenty
- hodnotu komponenty (napr. pre zdroj tam bude 10V)
- názov komponenty (napr. pre odpor R1)
- popis komponenty
- pozícia komponenty na mriežke

Element vodič má okrem svojho unikátneho čísla:

- zoznam bodov na mriežke, ktorými prechádza
- dva CIDy, teda objekty identifikujúce spojené komponenty

Element uzla má okrem unikátneho čísla:

- štyri PINy, teda vstupy do uzla (aby sme dodržali kolmost', sú povolené maximálne 4)
- názov uzla
- popis uzla
- pozícia na mriežke

PIN

PIN je štruktúra, ktorá popisuje vstupy a výstupy komponent. Teda konkrétne určuje, s ktorým prvkom je daný prvok spojený a ktorým vodičom.

Štruktúra PIN obsahuje:

- identifikáciu prvku, s ktorým je daný prvok spojený
- identifikáciu vodiča, ktorým je spojenie reprezentované

CID

CID je štruktúra, ktorá popisuje vodičom spojené komponenty. Jedna štruktúra CID opisuje jednu stranu spojenia, teda jednu spojenú komponentu. Obsahuje identifikáciu komponenty.

4 Implementácia

4.1 Platforma a vývojové prostredie

Systém TKSL a TKSL/C je momentálne najviac používaný na platforme Windows. Preto som zvolil práve Windows ako platformu pre grafický editor. Vývojové prostredie bežiacie pod Windows a spĺňajúce potreby pre čo najjednoduchšiu a najmodernejšiu implementáciu je podľa môjho názoru Microsoft Visual Studio 2005. Ako programovací jazyk som zvolil C# a technológiu .NET.

4.2 .NET

Je to zastrešujúci názov pre súbor technológií v softwarových produktoch, ktoré tvoria celú novú platformu, ktorá je dostupná pre Web, Windows a aj Pocket PC.

Pre tvorbu aplikácií splňujúcich tieto myšlienky vydal Microsoft Visual Studio .NET, ktoré bolo oproti predchádzajúcej verzii rozšírené o jednoduchý návrh webových XML služieb .NET, a .NET Framework, zaisťujúci prostredie potrebné pre beh aplikácií a ponúkajúci ako spúšťače rozhranie, tak potrebné knižnice, ako Java. Visual studio sa skladá z jazykov VB, J#, C# a Managed C++. Týmto jazykmi napísanú aplikáciu potom môžete bez problémov previesť do ASP.NET (Web) alebo do aplikácie pre pocket PC (.NET Compact Framework)

.NET Framework je pre majiteľa operačného systému Windows (vyžadovaná je minimálne verzia Windows 98) k dispozícii zdarma ako samostatný komponent, ktorý sa do systému doinštaluje (býva šírená na CD či DVD rôznych počítačových časopisov; dá sa taktiež stiahnuť cez Windows Update).

K svojmu behu ju vyžadujú väčšinou programy, ktoré pracujú s Active directory alebo SQL vďaka tomu, že .Net má tieto technológie dobre prístupné (rovnaký autor - Microsoft). .NET sľubuje aj lepšie grafické prostredie a operačný systém Windows Vista je na tejto technológii postavený.

K dispozícii je aj verzia .NET Compact Framework (.NET CF) pre Pocket PC s operačným systémom Windows Mobile, ktorá je s „klasickou“ verziou kompatibilná a tak nie je nutné kompilovať rôzne aplikácie pre PC a PDA - na oboch systémoch bude aplikácia fungovať rovnako. Ako už názov napovedá, .NET CF neobsahuje všetky objekty, funkcie a metódy z pôvodného .NET Frameworku.

GNU obdoba .NET sa nazýva DotGNU. Jej časť nazývaná DotGNU Portable.NET umožňuje spúšťať všetky .NET aplikácie na unixových platformách (Linux, BSD, Mac OS X, Solaris, AIX) a dokonca pomocou nástrojov Cygwin a Mingw32 aj na Windows.

V prostredí operačných systémov Linux, UNIX, Mac OS X je k dispozícii aj sada nástrojov kompatibilných priamo s Microsoft .NET pod názvom Mono. Túto sadu nástrojov však nevyvíja firma Microsoft, ale v obvyklom duchu opensource skupina dobrovoľných vývojárov.

Dve technológie často spojované s .NET sú ASP.NET a jazyk C#. [4]

4.3 Implementácia Databázy

Už v časti 3.3.3 bolo veľa napísané o forme akou je databáza písaná a prečo vlastne existuje. V tejto kapitole popíšem ako je konkrétne implementovaná.

Implementácia jednotlivých elementov je úplne rovnaká ako to bolo predložené v časti 3.3.3. Za dôležité považujem to, že celá trieda TDatabase je statická, to znamená, že sa vytvorí pri spustení aplikácie a existuje až do jej ukončenia. To je dôležité, pretože takto je prístupná všetkým modulom, ktoré patria do programu.

Všetky metódy, ktoré existujú nad Databázou sú tiež statické, to znamená, že nepotrebujú inštanciu triedy aby mohli byť volané. Databáza obsahuje len základné metódy a to Add (pridá element) a Delete (odstráni element). Databázu tvorí zoznam komponent, vodičov a uzlov. Všetky zoznamy sú public, teda sú viditeľné všetkým ostatným modulom. Tým pádom, operácie nad databázou si každý modul obsluhuje sám.

4.3.1 PIN

PIN je trieda, ktorá popisuje vstupy a výstupy komponent. Teda konkrétne určuje, s ktorým prvkom je daný prvok spojený a ktorým vodičom.

trieda PIN obsahuje:

- Identifikačné číslo prvku, s ktorým je daný prvok spojený
- Identifikačné číslo vodiča, ktorým je spojenie reprezentované
- Referenciu na objekt prvku
- Referenciu na objekt vodiča

Referencia je potrebná z dôvodu uľahčenia vyhľadávania. Je pravda, že každý prvok je určený podľa ID, ale je jednoduchšie a rýchlejšie uložiť konkrétnu referenciu na prvok, ako prehľadávať zoznam podľa ID. ID je dôležité pri ukladaní a načítaní vytvorených schém, pretože referenciu by sme asi ťažko uložili a následne načítali.

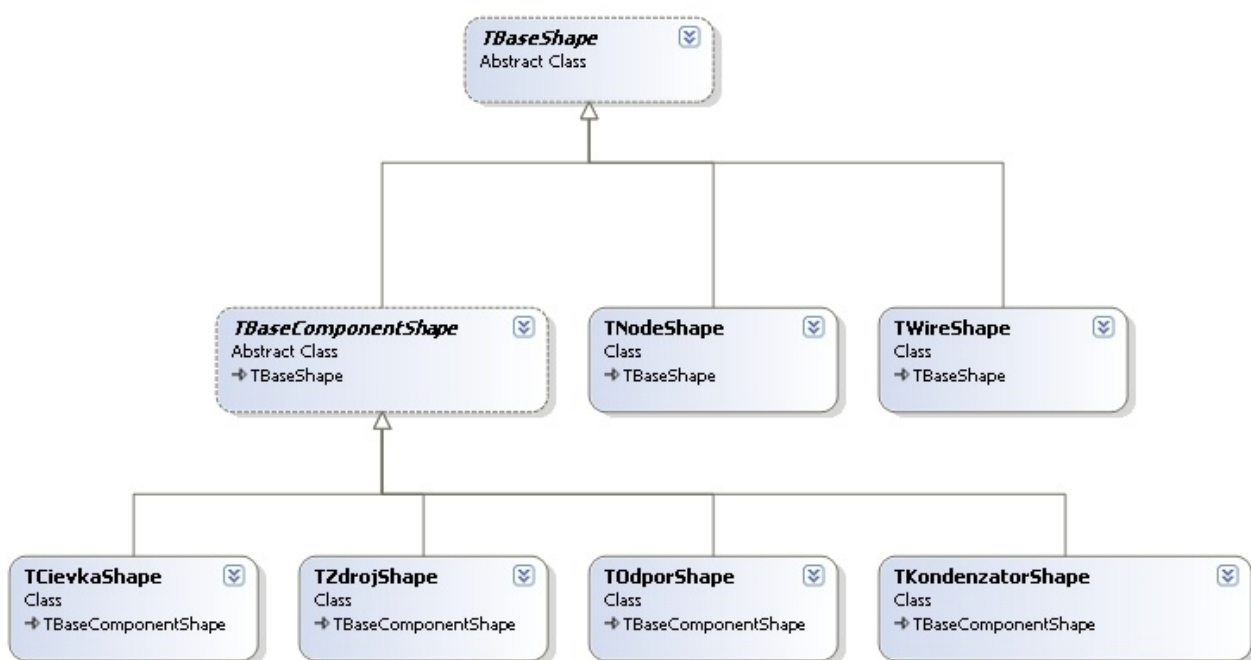
4.3.2 CID

CID je trieda, ktorá popisuje vodičom spojené komponenty. Jedna štruktúra CID opisuje jednu stranu spojenia, teda jednu spojenú komponentu. CID obsahuje ID komponenty a referenciu na ňu. Dôvod je analogický s dôvodom v PIN (časť 4.3.1)

4.4 Implementácia vykresľovaných tvarov

Pod pojmom vykresľovaný tvar rozumieme komponentu, vodič a uzol. Teda všetko to, čo sa môže objaviť na plátne.

Pri vykresľovaní týchto tvarov je pre programátora veľmi výhodná dedičnosť. Tvary objektov v našom prípade majú spoločnú triedu *TBaseShape*. Od triedy *TBaseShape* sú podedené triedy *TWireShape*, *TNodeShape* a *TBaseComponentShape*. A nakoniec od triedy *TBaseComponentShape* sú podedené triedy *TKondenzatorShape*, *TCievkaShape*, *TOdporShape* a *TZdrojShape*. Celú dedičnosť ukazuje obrázok 4.4.



Obrázok 4.4: Dedičnosť vykresľovaných tvarov

Pri znalosti dedičnosti nám musí byť jasné, že ktorýkoľvek objekt akejkoľvek triedy, môžeme uložiť do kolekcie¹ prvkov triedy *TBaseShape*. A presne tak je riešené vykresľovanie. Pri nutnosti vykresliť plátno sa prejde celou kolekciami a na každý objekt v kolekcii sa zavolá jeho metóda *DrawMe()*. *DrawMe* je virtuálna metóda, ktorú si nadefinuje každá trieda podľa seba. To znamená, že cievka sa bude vykresľovať inak ako kondenzátor a podobne. Ale v samotnom prechádzaní kolekciami sa bude volať iba jedna metóda a každý objekt zavolá svoju špecifickú *DrawMe*. Týmto docielime programátorský komfort a veľmi flexibilné pridávanie ďalších komponent.

Základná trieda obsahuje oveľa viac virtuálnych metód, ale nebudeme sa tu všetkými zaoberať. Všetky sú okomentované v zdrojových súboroch.

¹ v našom prípade je to zoznam

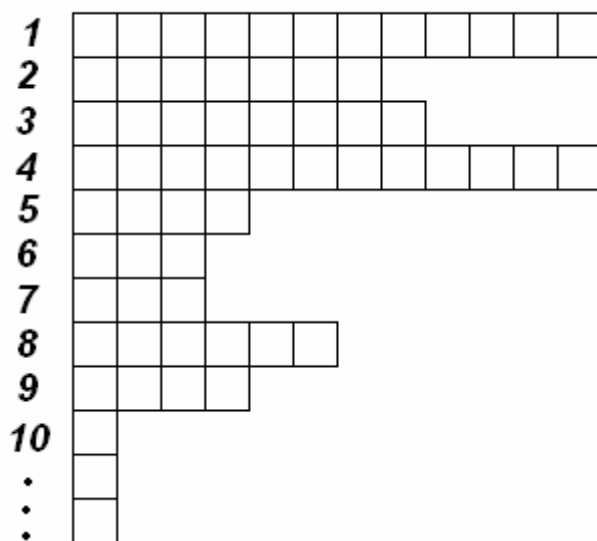
Je treba spomenúť, že každá listová trieda (trieda od ktorej už žiadna iná nededí) obsahuje referenciu na objekt v databázy. To znamená, že pri vytvorení objektu na plátne sa vytvorí objekt tvaru a vloží sa do kolekcie všetkých tvarov, ale zároveň sa vytvorí objekt toho daného typu aj v databázy a referencia naň sa vloží do objektu tvaru. Takže každý objekt tvaru bude mať svoje údaje uložené v databázy a prístup k nim bude cez referenciu.

Základný objekt obsahuje dve dôležité virtuálne metódy `AddToGrid` a `DeleteFromGrid`. Tieto metódy, ako názov hovorí uložia body komponenty, vodiča, či uzla do mriežky a to tak ako to konkrétna komponenta, vodič alebo uzol vyžadujú.

4.5 Implementácia mriežky

Mriežka teda slúži ako prostriedok na väčší komfort užívateľa. Ale na druhú stranu sa dá využiť aj na účely rýchleho získania informácií o aktuálnej pozícii kurzora myši. Vždy keď kurzor prechádza ponad plátno, nachádza sa nad určitým bodom mriežky a tento bod obsahuje informácie o objekte, ktorý sa na ňom nachádza. Je to jednoduchšie a rýchlejšie ako vždy podľa súradníc prechádzať všetky objekty a zisťovať nad ktorým práve sme.

Mriežka je implementovaná ako riedka matica. Tým pádom nezaberá toľko pamäťových prostriedkov ako normálna matica $N \times N$. Riedka matica je implementovaná ako pole zoznamov položiek mriežky. Teda vlastne y-ová súradnica je pevne daná veľkosťou poľa a x-ová súradnica môže mať ľubovoľnú veľkosť. Problémom je len to, že každá položka v zozname musí obsahovať parameter index, ktorý určuje x-ovú súradnicu.

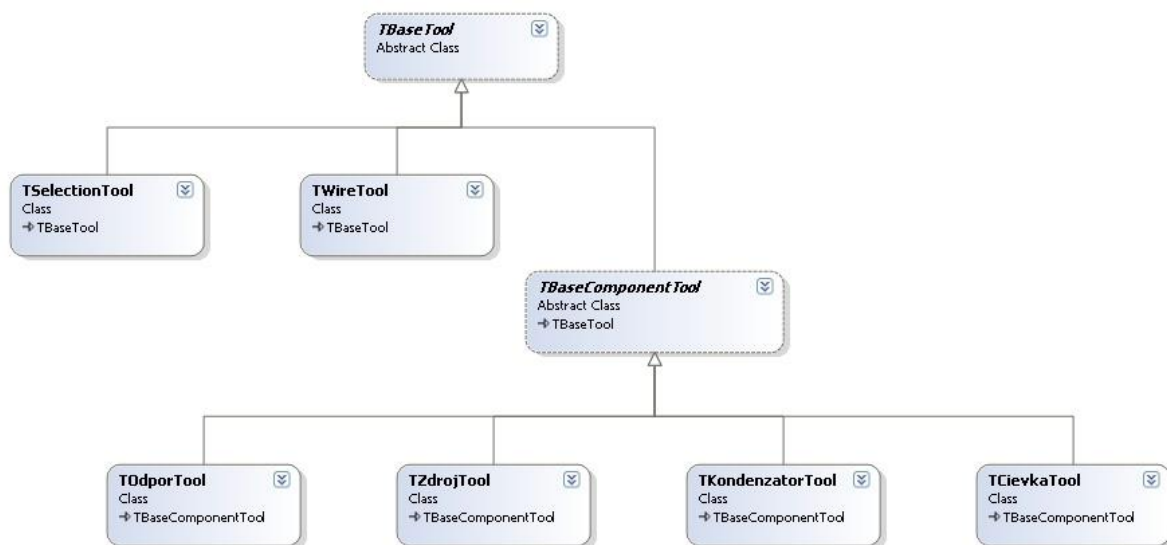


Obrázok 4.5: Mriežka – riedka matica

4.6 Implementácia kresliacich nástrojov

Každý tvar, ktorý chceme vykresliť má svoj kresliaci nástroj, teda triedu, ktorá pri určitých okolnostiach vytvorí zadaný objekt. Aj pri týchto triedach je veľmi pružné a prezieravé použiť dedičnosť. Pri jej použití je potom jednoduché vytvoriť jeden objekt základnej triedy a volať jeho virtuálne metódy pri určitých udalostiach. A záleží aký objekt (akého typu) je momentálne v tomto objekte základnej triedy uložený. Pretože ak tam bude objekt kreslenia cievky, bude sa kresliť cievka.

Dedičnosť je navrhnutá analogicky k vykresľovaným tvarom. Zobrazuje ju obrázok 4.6.



Obrázok 4.6: Dedičnosť kresliacich nástrojov

Základná trieda obsahuje virtuálne metódy MouseDown, MouseMove, MouseUp. Tieto metódy pre každú podedenú triedu vyjadrujú, čo sa stane ak nastane udalosť od myši. A tiež obsahuje virtuálnu metódu DeleteActualObject, ktorá pri nástroji selection odstráni aktuálny objekt.

Pri kliknutí na nejaký bod plátna sa tento bod prepočíta do mriežky a zistí sa na aký objekt sme klikli. Podľa tohto objektu sa konkrétna metóda zariadi. Napríklad ak ťaháme vodič a klikneme na prázdnu plochu, vytvorí sa ďalší segment vodiča a ak je to nutné tak sa zalomí. Bod zalomenia sa vypočíta ako bolo spomenuté v časti 3.3.2. Ak sa klikne na vstup alebo výstup komponenty, tak sa kreslenie ukončí a vodič sa uzavrie a pripojí na obe spájajúce komponenty.

Obsluha z hlavnej triedy TCanvas je teda veľmi jednoduchá, pri udalosti myši sa zavolá konkrétna metóda na spracovanie tejto udalosti. Ďalšie metódy v TCanvas sú na nastavenie nástroja, prípadne vymazanie objektu.

5 Použitie

5.1 Zapojenie modulu do programu

Kresiaci modul sa veľmi jednoducho zapojí do iného programu. Celý modul je autonómny a jediné, čo potrebuje je vytvoriť v programe inštanciu triedy TCanvas. Potom už len pri vzniknutých udalostiach myši volať príslušné metódy. A je jasné, že ak budeme chcieť zmeniť kresliaci nástroj tak zavoláme konkrétnu funkciu, ktorá nám zmení nástroj na požadovaný.

Metódy na spracovanie udalostí myši vyžadujú jeden parameter a to parameter konkrétnej udalosti myši². Posledná nutná vec je v metóde na prekreslenie³ zavolať metódu na prekreslenie plátna – Paint, ktorá vyžaduje parameter udalosti kreslenia, teda PaintEventArgs.

5.2 Manuál

Program priložený k práci má jednoduché GUI, ktorým sa dá kresliaci modul ovládať. Pri kliknutí na tlačítko cievky sa vyberie kresliaci nástroj cievka. Pri kliknutí na tlačítko vodiča sa bude vykresľovať vodič a analogicky zvyšok tlačítiek.

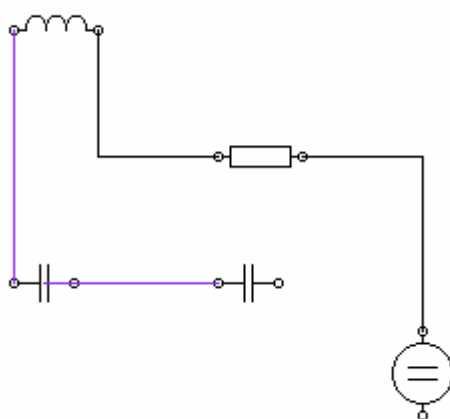
Pri zvolení cievky sa pri stlačení myšieho tlačítka objaví na plátne obraz cievky, ktorý môžeme ľubovoľne presúvať po ploche kým držíme tlačítko. Pri pustení tlačítka sa cievka uloží na nami určené miesto.

Ak chceme umiestniť druhú cievku alebo iný prvok tak aby bol na jednej línii s iným prvkom, pomôžu nám k tomu zarovnávacie čiary. Sú to fialové čiary, ktoré sa nám zobrazujú pri umiestňovaní prvkov vtedy, keď ležia na rovnakej línii. To znamená, že ak kreslíme kondenzátor a už existuje iný prvok na plátne, tak ak sú v jednu chvíľu vstupy alebo výstupy niektorých dvoch prvkov na rovnakej línii (je jedno, či podľa stĺpca alebo riadku) tak sa nám zobrazí pomocná fialová čiara, ktorá nás ubezpečí, že máme prvok na dobrom mieste.

Pri kreslení vodičov je nutné začať kresliť v bode vstupu/výstupu komponenty. Vstupy a výstupy sú znázornené ako malé kružnice na konci a začiatku prvku. Pri kliknutí na túto kružnicu začneme kresliť vodič (je nutné vybrať nástroj kreslenia vodiča). Pri kreslení vodiča nie je nutné aby sme držali tlačítko myši stlačené. Reakcie kreslenia sú závislé na samostatnom kliknutí. Kreslenie vodiča sa ukončí pri kliknutí na iný (ešte neobsadený) vstup/výstup. Vodič sa pri kreslení zalamuje.

² MouseEventArgs

³ Udalosť Paint



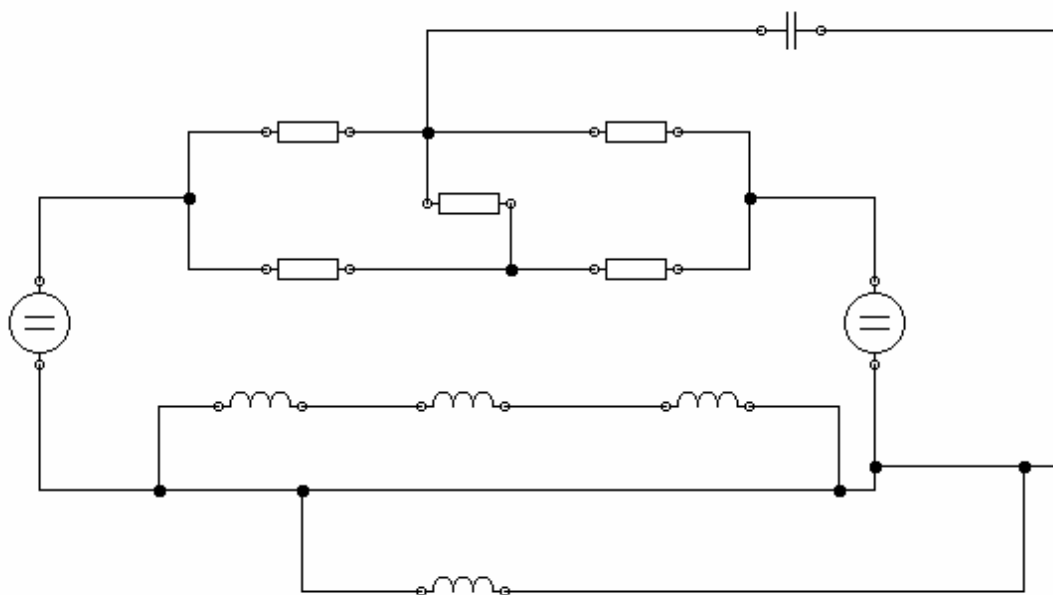
Obrázok 5.2a: zarovňavacie čiary

Pri voľbe nástroja Selection sme schopný označiť a presunúť existujúce objekty, dokonca aj tie ktoré sú už zapojené. A pri označenom prvku sa dá po stlačení tlačítka delete prvok odstrániť. Označiť sa dá všetko od komponent cez vodiče po uzly. Ale pohybovať sa dá zatiaľ iba s komponentami.

Pri kreslení vodiča môžeme koniec vodiča pripojiť na už existujúci vodič a tak automaticky vytvoriť uzol. Do uzla môžu viesť maximálne 4 vodiče, viac vám program nedovolí.

Pri odstránení komponenty sa odstránia aj vodiče do nej vedúce a pri odstránení uzla sa odstránia taktiež všetky vodiče, ktoré sa v ňom spájajú.

V takomto kresliacom programe sa dá nakresliť aj nie moc triviálne schéma (Obrázok 5.2b).



Obrázok 5.2b: Schéma nakreslená v kresliacom programe

6 Záver

Cieľom tejto práce bolo vysvetliť možnosti riešenia elektrických obvodov a to hlavne pomocou diferenciálnych rovníc. Ďalším bodom bolo poukázať na problémy pri tvorbe diferenciálnych rovníc a vysvetliť teóriu parazitných kondenzátorov, ako riešenie mnohých problémov s výpočtom.

Druhým krokom bolo navrhnuť grafickú nadstavbu nad systémom TKSL, v ktorej by bol užívateľ schopný nakresliť schéma bez väčších komplikácií. Základné funkčné črty kresliaceho modulu boli navrhnuté a implementované. Vývoj ešte nie je ukončený a existuje zoznam známych nedostatkov, ktoré sa budú riešiť v najbližšom čase.

Veľký prínos práce spočíval v bližšom pochopení diferenciálnych rovníc a samozrejme s ich použitím pri riešení elektrických obvodov. Druhým prínosom bolo prehĺbenie znalostí o prostredí Visual Studio a platforme .NET.

Na editore bude potrebné doladiť neošetrené výnimky a hlavne vytvoriť pekné grafické rozhranie, ktoré bude plne podporovať už známe funkcie editora. Ale napriek týmto známym nedostatkom je editor plne funkčný a dá sa použiť na kreslenie jednoduchých ale aj zložitých schém. Jeho hlavná črta je oddelenie kresliaceho modulu od vonkajších modulov ako je GUI a samotný transformačný model, takže je jednoducho možné ho zakomponovať do viacerých projektov.

Literatúra

- [1] Jiří Petřek. Diplomová práce. Brno, VUT, 2001
- [2] Milan Tichý. *Elektronické zpracování signálů*. Karolinum, nakladatelství UK, Praha 1998.
- [3] Jiří Konvalina. Datové vstupy simulacního systému TKSL. Brno, VUT, 2006
- [4] *Wikipedie: Otevřená encyklopedie: .NET* [online]. c2007 [citováno 8. 05. 2007]. Dostupný z WWW: <<http://cs.wikipedia.org/w/index.php?title=.NET&oldid=1412460>>

Príloha A

Obsah priloženého CD

Na priloženom CD sa nachádzajú tieto adresáre:

\DOC\

Obsahuje túto prácu vo forme PDF (adresár PDF) a vo forme DOC (adresár DOC), ďalej obsahuje v podadresári PICTURES použité obrázky pri tvorbe tejto práce. V podadresári XML je XML dokumentácia zdrojových súborov.

\EDITOR\

Obsahuje spustiteľný súbor editora.

\EDITOR-SOURCE\

Obsahuje zdrojové kódy kresliaceho modulu aj s vytvoreným projektom vo Visual Studiu 2005.